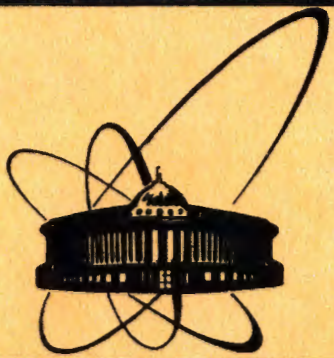

Сообщения
Объединенного
института
ядерных
исследований
дубна

E10-85-541

J.Schenk*, P.Wegner*, M.Winde

THE COSMIC PROGRAM SYSTEM FOR FLEXIBLE PROGRAMMING IN ON-LINE EXPERIMENTS

* Institut für Hochenergiephysik
der AdW der DDR, DDR-1615, Berlin-Zeuthen

1985

1.0. INTRODUCTION

When writing programs for scientific experiments, especially for on-line data acquisition, the detailed analysis of problems is typically not completed in advance. This demands for flexible programs/1/. The COSMIC system supports flexible programming in on-line experiment data acquisiton.

The COSMIC system/2/ runs on PDP-11-like computers under control of the RSX-11M or RT-11 operating systems.

COSMIC provides a simple and very flexible language COSMICRO to perform essential steps of program design, coding and execution control by commands uniformly.

COSMICRO:

- supports top-down design;
- allows programming of the user program until a certain level by stepwise refinement. Below that level the user program is coded in FORTRAN subroutines (the so-called "action routines") which are called from COSMICRO. There are action routines for many typical tasks programmed in advance;
- is used as the command language of the user program;
- provides means for user program test at the source language level;
- provides means for program maintenance.

The COSMIC system provides dynamic management of memory for data. It is highly overlayed. Thus it is possible to perform even bigger tasks with few memory (min. 20 K). Using COSMICRO dramatically reduces the development time for many common tasks in experiment automation.

The COSMICRO user program part is executed by an interpreter. This results in execution times considerably higher compared with programs written in FORTRAN.

2.0. SOLVING A PROBLEM WITH COSMIC

A simple example demonstrates how COSMIC principially works. The values of two analogue-to-digital converters (ADC) connected to the computer via CAMAC are to be read out. The values are filled into two histograms. The histograms are presented to the user on a display or a printer optionally.

All programs are developed top-down and stepwise refined using the COSMICRO language. Our problem may be divided into three steps:

- initialisation UNIT
- data taking TAKE
- presentation SHOW

We denote each of the steps by an alphanumeric string, the so-called "micro".

```
RUN = INIT TAKE SHOW
```

What must be done in the steps in detail is unknown for the time being. It has to be refined now. In the initialisation step information units of the dynamically manages memory area must be booked for storing data and histograms. The ADC must be initiated. We use micros, once more.

```
UNIT = BANKBOOK HISTBOOK CAMACINIT
```

In the data taking step the ADCs must be read out. The values must be filled into the histograms. The following has to be performed:

- wait for a signal of the trigger WAITTRIGGER
- read-out of ADCs READADC
- histogram values HISTVALUES
- clear ADCs CLEARADC

A number of events shall be executed, i.e., the process is to be performed NTAKE times. We get:

```
TAKE = NTAKE [EVENT]
EVENT = WAITTRIGGER READADC HISTVALUE CLEARADC
```

The program is very general until now. Even the number of ADCs to be treated is not specified. This is done now:

```
READADC = READADC1 READADC2
CLEARADC = CLEARADC1 CLEARADC2
HISTVALUES = HIST1 HIST2
SHOW = SHOW1 SHOW2
```

If further refinement in COSMICRO is unwise or even impossible the micros undefined until now are defined as action routine calls:

```
BANKBOOK = IBOOK(2,10)
```

(a bank in the dynamic storage area for storing the ADC values is booked).

```
HISTBOOK = H1BOOK(101,BIN1, TITLE1) H1BOOK(102,BIN2,TITLE2)
```

(Two histograms are booked).

The treatment of the ADCs is done by two action routines.

Routine ADCINI initiates an ADC:

```
CAMACINIT = ADCINI(NADC1) ADCINI(NADC2)
```

Further treatment is by routine ADC, which may be called with a variable number of parameters:

```
WAITTRIGGER = ADC (WAIT, NADC1)
```

(as both ADCs will be ready for read out on the same trigger signal, the program must wait only for one of them)

```
READADC1 = ADC(READ, NADC1,#ADC1)
READADC2 = ADC(READ, NADC2,#ADC2)

CLEARADC1 = ADC(CLEAR, NADC1)
CLEARADC2 = ADC(CLEAR, NADC2)
```

Filling and representation of historgams are defined:

```
HISTV1 = H1FILL(101, @ADC1)
HISTV2 = H1FILL(102, @ADC2)

SHOW1 = H1SHOW(101, DEVICE)
SHOW2 = H1SHOW(102, DEVICE)
```

All action routines are defined now. It remains to define the variables used.

We will process 100 value pairs, for the moment:

```
NTAKE = 100
```

Number of bins, lower bound and bin width of the histograms must be defined, the titles have to be defined:

```
BIN1 = 100,0,10
BIN2 = 100,0,10

TITLE1 = 'SEV1, HV=1.8 kV'
TITLE2 = 'SEV2, HV=1.8 kV'
```

The CAMAC station numbers of the ADCs are fixed:

```
NADC1 = 12
NADC2 = 13
```

Locations for storing the ADC values must be fixed, they are in the bank:

```
ADC1 = 2.1
ADC2 = 2.2
```

The devices foreseen for histogram presentation must be specified:

```
DEVICE = DISPLAY
DISPLAY = 5
PRINTER = 6
```

Last we have to define the function identifiers WAIT, READ and CLEAR of action routine ADC in a form suitable for passing them to the FORTRAN written action routine. We use integers:

```
WAIT = 1
READ = 2
CLEAR = 3
```

At this point we have defined the necessary action routines. Also their interface to the COSMICRO program part is established. We are going to code the action routines.

In our example some very general action routines are used. We can take them out of the library of action routines programmed in advance. These are IBOOK, H1BOOK, H1FILL, H1SHOW. Routines ADCINI and ADC control the work of a given CAMAC module in a general way. If that module was used with COSMIC before, these routines probably exist. Otherwise they must be coded, then compiled.

When all action routines exist in object code form, we define an overlay structure and write it to a disc file. We also write the micro definitions to a disc file. The figure summarizes all definitions in an easy to survey format (as they should be written to the file).

Using this information a special service program links the runable COSMIC program (with the help of the linker of the operating system used). The program is started.

Now we can control the run of the user program in a very flexible way by typing commands to the console terminal. To form commands all defined micros or other COSMICRO language elements may be used.

If the command RUN is typed, e.g., it is first translated (i.e., expanded and somewhat compacted) then executed. The micro apparatus proceeds the expansion exactly as the human user just now when developing the program.

The micro apparatus finds the elementary component INIT, to begin with. INIT is defined as a micro. Thus it is expanded according to its definition giving BANKBOOK HISTBOOK CAMACINIT. Next BANKBOOK is found, identified as a micro and expanded to IBOOK(2,10). This expression does not contain any further micro, thus it is taken over unchanged. Expansion continues with HISTBOOK, etc., until the whole command is expanded.

Afterwards the expanded expression is somewhat compressed then executed by an interpreter. If errors are found during translation or execution, they will be messaged to the user via the terminal display.

In our example no errors were found. The histograms presented seem to be all right, but we need a number of events in addition. We type the commands

```
NTAKE = 900
TAKE SHOW
```



```
RUN = INIT TAKE SHOW

INIT = BANKBOOK HISTBOOK BANKBOOK CAMACINIT
HISTBOOK = H1BOOK(101,BIN1,TITLE1) H1BOOK(102,BIN2,TITLE2)
CAMACINIT = ADCINI(NADC1) ADCINI (NADC2)
BANKBOOK = IBOOK(2,10)

TAKE = NTAKE [EVENT]
     EVENT = WAITTRIGGER READADC HISTVALUE CLEARADC
        WAITTRIGGER = ADC (WAIT, NADC1)
        READADC = READADC1 READADC2
                READADC1 = ADC (READ,,NADC1, #ADC1)
                READADC2 = ADC (READ, NADC2, #ADC2)
        HISTVALUE = HISTV1 HISTV2
                  HISTV1 = H1FILL (101, @ADC1)
                  HISTV2 = H1FILL (102, @ADC2)
        CLEARADC = CLEARADC1  CLEARADC2
                 CLEARADC1 = ADC (CLEAR, NADC1)
                 CLEARADC2 = ADC (CLEAR, NADC2)
SHOW = SHOW1 SHOW2
     SHOW1 = H1SHOW (101, DEVICE)
     SHOW2 = H1SHOW (102, DEVICE)
NTAKE = 100
     BIN1 = 100,0,10
     BIN2 = 100,0,10
     TITLE1 = 'SEV1, HV=1.8 kV'
     TITLE2 = 'SEV2, HV=1.8 kV'
              NADC1 =12
              NADC2 =13
              ADC1 = 2.1
              ADC2 = 2.2
                       WAIT=1
                       READ=2
                       CLEAR=3
     DEVICE = DISPLAY
     DISPLAY = 5
     PRINTER = 6
```

Example of a COSMICRO program.

and get histograms with 1000 events at all. The same result we would have got by typing

```
900 [EVENT] SHOW
```

or also by

```
9 [TAKE] SHOW
```

Which of the possible commands is actually typed by the user, typically depends on that, which one comes to his mind first or, more seldomly, by which one the task to be performed can be expressed most compactly.

Other possible actions: The histogram of ADC 1 values shall be presented on the printer. This may be done by typing

```
DEVICE = PRINTER
SHOW1
```

or, alternatively by

```
H1SHOW(101,PRINTER)
```

Now we change a hardware parameter. We note this in the histogram title:

```
TITLE1 = 'SEV1, HV=1.35 kV'
```

Now we start a new data taking run. We want to get 10.000 events. After every 200 events we want to have presented the histogram of ADC 1 on the display. At the end of the run we want both histograms on the printer:

```
DEVICE = PRINTER
DIS = H1SHOW(101,DISPLAY)
TAKE = 50 [200 [EVENT] DIS] SHOW
RUN
```

This example gave a glance how COSMICRO works. All demands could be fulfilled without modifying action routines, i.e., without time wasting compiler and linker runs. The user simply typed the commands at the console, the system executed them instantly.

Data already taken stayed disposable after performing program modifications, if required.

The flexibility of the program was not exhausted in our example. Other usefull modifications may be:
changing the definition area of a histogram:

```
BIN1 = 80, 40, 2
```

using a different ADC:

```
NADC1 = 18
```


3.0. LANGUAGE ELEMENTS FOR DESIGN, CODING AND COMMANDS

In detail there are the following elements for design, coding and commands:

1. Micros

The user may define text strings of arbitrary length and content as micros

```
micro-name = micro-expansion
```

"Micro-names" are arbitrary alphanumeric strings. During translation every "micro-name" is replaced by its "micro-expansion". Actions, sequences of actions, parameters, etc., may be denoted by micros. Micros are nestable.

Example: READ = VISUALIZE THINK

2. Repetitions

```
repeat-number[actions]
```

The "actions" are executed "repeat-number" times.

Example: 2 [READ]

The "actions" may contain expressions of the form

```
WHEN logical-variable EXIT
```

at possibly several passages. Each of them causes termination of the repetition if the "logical-variable" yields the value true. If the "repeat-number" is omitted, the "actions" are performed an unlimit number of times. Those repetitions are terminated by WHEN-EXIT expressions only.

Example: [READ WHEN CAUGHT EXIT]

3. Branches

```
CASE variable OF
case1: actions1
case2: actions2
...
caseN: actionsN
ENDCASE
```

The value of "variable" is compared with the value of "case1". If they are equal, "actions1" are executed. Next the value of "variable" is compared with the value of "case2". If they are equal "actions2" are executed, etc. Finally the value of "variable" is compared with the value of "caseN". If they are equal "actionsN" are executed.

As the "cases" may be variables too, zero, one or more "actions" may be executed.

```
Example:  CASE CAUGHT OF
          TRUE: THINK-AHEAD
          FALSE: READ
          UNCERTAIN: AQXBR (READ, 3, "24, ASK)
          ENDCASE
```

4. Action Routine Calls

actionroutinename (parameter1,...parameterN)

A FORTRAN coded part of the user program (a subroutine) is going to be executed. The parameters are submitted to the action routine.

There are several possibilities to denote a variable (repeat-number, logical-variable, variable, case, parameter). Variables may be

1. decimal or octal integer constants, real constants
   (examples: 10, "12, 10.000)
2. addresses or contents of bank elements
   (examples: #2.1, @2.1)
3. text strings
   (example: 'THIS IS A STRING')

Variables or parts of variables may be denoted as micros (examples: "OCTALVALUE, @BANK.3)

4.0. MEANS TO SUPPORT THE TEST

By the use of tracing commands the user asks COSMIC to inform him about the succession of action routine executions. Some or all action routines can be selected for tracing. A message is given every time an action routine selected for tracing is executed.

Furthermore the test is facilitated by error recognition and messages by the COSMIC system during translation (syntax errors) or program execution (wrong use of information units, wrong number of parameters of an action routine call).

COSMIC supports the bottom-up test. It is a good practice first to test the function of single action routines by typing the corresponding commands, then to test the cooperation of those action routines united to micros at the lowest level, etc., until the command is typed that will finally start the whole process.

Special test micros can be included in some micro definitions too. They may be defined as empty after all tests are fulfilled. Thus they are without effect and even do not waste execution time then.

These simple but powerfull means provide the test of the COSMICRO user program part on the source language level.

5.0. INTERVENTIONS

The user has two possibilities to intervene when a user program runs:

1. By typing the command ECR (End COSMIC Run) execution of the actual command string is aborted after the currently active action routine has finished. COSMIC asks for the next command. All information units (banks, histograms) and their contents remain unchanged. Defined micros remain.
2. By typing the command ICR (Interrupt COSMIC Run) execution of the actual command string is interrupted after the currently active action routine has finished. COSMIC asks for a command to perform the so-called "foreground action". Information units booked in the background may be used by the foreground action.

Commands for foreground actions have the same syntax as commands for background actions. All micros defined are at the user's disposal. After the end of the foreground action the background action continues immediately.

Foreground actions are typically used to present some data or histograms "on the fly", sometimes also to change some variables. It is possible to change tracing conditions as a foreground action, thus allowing especially flexible testing of the background program.

6.0. CRITICISM OF THE SYSTEM

Using COSMIC spares about 50 to 80% of program development time. Great flexibility is provided to the user by the ease of utilization and modification of the user programs. This is especially usefull in on-line experiments, where new questions arise often.

The syntax of the COSMICRO language should be extended. Most worth mentioning is the lack of arithmetical and logical expressions. Flexibility can be further increased by passing micros (without parameters) to macros (with parameters) or similar more powerfull expansion techniques.

The most important weakness of the system is the low execution time efficiency of the programs. Execution time of a program is typically 20 to 500% higher compared with programs written entirely in FORTRAN. For most applications this is not crucial. On the other hand there were real time applications where one had to find a compromise to the debit of flexibility. For that purpose some parts of the user program originally written

in COSMICRO had to be re-written as FORTRAN action routines. To overcome this weakness we intent to include a COSMICRO-compiler into the system. The compiler will generate machine code programs thus giving maximal efficiency.

Another important weakness is the necessity to work with two languages to achieve flexibility (COSMICRO) and to denote the details of the program (FORTRAN). Obviously it would be very much better to use a uniform language for the whole user program. Such a uniform language or its compiler, respectively, muct provide the following attributes:

1. The syntax of the language must provide a maximum of the syntax elements known from modern high level language (program modules, abstract data types, elements for structured coding, probably multi-tasking possibilities). On the other hand there must be a subset of the language that can be used easily and quickly in order to meet its function as a command language. Finally it must be easy to learn and easy to use (at least there must be an easy subset) because it will be used especially by scientists not being software engineers.
2. The language must provide possibilities for interactive influence to program continuation. Especially there must be commands to present data "on the fly".
3. There should be a dynamic memory management.
4. There must be a compiler that generates effective programs.

None of the widely spread languages fulfills all of these demands. For that, systems are necessary which integrate compilers, interpretes and linkers. One of several ways may be gone:

System A

The system includes a compiler-linker and an interpreter. The user selects the program parts to be compiled.

System B

A compiler translates all smallest defined program components (in COSMIC these would be all micros). Every time a command was typed the linker forms a runable program out of the components used by that command.

System C

The program is completely compiled and linked. The table of "global" defined symbols remains at disposal after that. Every command typed is interpreted until an entry point into the compiled program is found. From that the compiled program is executed until it returns to that entry point. Then interpretation continues, etc.

Each of the proposed systems has superiorities as well as drawbacks if compared with the others. System A calls the user for additional decisions in situations often hardly to decide. System B will need long commandexecution times because the linker must run after every command. System C reacts promptly on commands, but the whole program must be linked another time after every program modification.

7.0. CONCLUSION

The COSMIC program system for experiment automation provides for flexible programming. Its most important weakeness are the low efficiency of programs and the necessity to program in two distinct languages. Better efficiency will be achieved by the construction of a COSMICRO-compiler in near future. It should be possible to overcome the two-languages-problem by defining a language that is a command language as well as a general program language. The programs must be executed by a compiler-linker-interpreter system to be constructed.

The authors want to express their acknowledgement to Prof. H.Schiller for his permanent interest to the investigations underlying this paper.

REFERENCES

1. Winde M. Experiment Automation Demands for Flexible Programs. JINR, E10-85-540, Dubna, 1985.
2. Wegner P., Winde M. COSMIC-Programmdokumentation. Institut fur Hochenergiephysik der AdW der DDR, Berlin-Zeuthen, 1983.

Received by Publishing Department
on July 12, 1985.

COMMUNICATIONS, JINR RAPID COMMUNICATIONS, PREPRINTS, AND PROCEEDINGS OF THE CONFERENCES PUBLISHED BY THE JOINT INSTITUTE FOR NUCLEAR RESEARCH HAVE THE STATUS OF OFFICIAL PUBLICATIONS.

JINR Communication and Preprint references should contain:

- names and initials of authors,
- abbreviated name of the Institute (JINR) and publication index,
- location of publisher (Dubna),
- year of publication
- page number (if necessary).

For example:

1. *Pervushin V.N. et al. JINR, P2-84-649, Dubna, 1984.*

References to concrete articles, included into the Proceedings, should contain

- names and initials of authors,
- title of Proceedings, introduced by word "In:"
- abbreviated name of the Institute (JINR) and publication index,
- location of publisher (Dubna),
- year of publication,
- page number.

For example:

Kolpakov I.F. In: XI Intern. Symposium on Nuclear Electronics, JINR, D13-84-53, Dubna, 1984, p.26.

Savin I.A., Smirnov G.I. In: JINR Rapid Communications, N2-84, Dubna, 1984, p.3.

Шенк Ю., Вегнер П., Винде М. E10-85-541
Система программирования COSMIC
для гибкого программирования в экспериментах на линии с ЭВМ

При разработке программ для научных экспериментов, особенно для on-line сбора данных, как правило невозможно предусмотреть все детали заранее. Поэтому программы должны быть гибкими. Система программирования COSMIC обеспечивает гибкое программирование для задач сбора данных в процессе эксперимента. Особенностью системы является унифицированный язык COSMICRO, обеспечивающий разработки и кодирования программ; управления во время сбора данных, которые можно перестраивать, не прерывая процесса их набора; отладку на уровне исходного текста. В статье на конкретном примере показан принцип работы системы COSMIC, описаны все элементы языка COSMICRO. Показаны также некоторые недостатки системы и направления ее развития.

Работа выполнена в Отделе новых методов ускорения ОИЯИ.

Сообщение Объединенного института ядерных исследований. Дубна 1985

Schenk J., Wegner P., Winde M. E10-85-541
The COSMIC Program System
for Flexible Programming in On-Line Experiments

When writing programs for scientific experiments, especially for on-line data acquisition, the detailed analysis of problems is typically not completed in advance. That demands for flexible programs. The COSMIC program system supports flexible programming in on-line experiment data acquisition. A uniform language for parts of design, coding and run time commands is provided. It provides the changing of the user program at data-take time. Program testing at the source language level is supported. The paper gives an introduction to the principal working scheme of the COSMIC program system by discussing an example. The summary of the elements of the language COSMICRO for design coding and run time commands for COSMIC user programs is given. The paper includes a criticism of the system and discusses future developments.

The investigation has been performed at the Department of New Acceleration Methods, JINR.

Communication of the Joint Institute for Nuclear Research. Dubna 1985